

## 1. Environment

- A. OS : Windows 10
- B. compiler version : TDM-GCC 4.9.2 64-bit
- C. IDE : Dev-C++ 5.11

## 2. Methods or solutions

### A. Red-Black Tree 宣告方式

- i. 各節點宣告
  - 1. 以 struct 宣告，命名為 node
  - 2. 每一元素都有 node 型別的指標變數 Leftchild、Rightchild、parent 各一
  - 3. 每一元素有一 int 型別的變數 NodeValue 存取該元素數值，
  - 4. bool 型別的變數存取顏色：true 表紅色，false 為黑色
- ii. Red-Black Tree 架構
  - 1. 有一 node 型別的指標 root 指向樹的根節點：即沒有 parent 者
  - 2. 有一 node 型別的指標 NIL 作為空節點：空節點的顏色必為黑色

### B. Insert 函式

- i. 需傳入要新增的節點的值(NodeValue)
- ii. 將新節點的顏色設為紅色，且左子及右子皆指向 NIL
- iii. 尋找新節點插入位置
  - 1. 若根節點為空，則新節點即為根節點，並塗成黑色
  - 2. 若根節點不為空
    - 甲、若新節點的值小於該節點，即繼續與該節點的左子比較
    - 乙、若新節點的值大於或等於該點，即繼續與該節點的右子比較
    - 丙、不斷重複上述動作，直到比較對象為 NIL，將新節點的 parent 指向前一個比較對象，且根據新節點的值與 parent 的比較結果，決定新節點是左子或是右子
    - 丁、若我的 parent 也是紅色即進入 insert\_fixup 函式：因違反紅黑樹規則，紅色需與黑色相連，不可與紅色相連

### C. Insert\_fixup

- i. 需傳入要修正的節點

- ii. 將情況分為該修正節點是他的 parent 的左子及該修正節點是他的 parent 的右子，且此兩種情況都各有三個 case
- iii. 修正節點的 parent 是左子
  - 1. Uncle 是紅色
    - 甲、將 Uncle 及 Parent 塗成黑色
    - 乙、Grandparent 塗成紅色
    - 丙、將修正節點改為 Granparent
  - 2. Uncle 是黑色且修正節點為其 parent 的左子
    - 甲、將 Parent 塗成黑色
    - 乙、Grandparent 塗成紅色
    - 丙、對 Grandparent 做一次右旋
  - 3. Uncle 是黑色且修正節點為其 parent 的右子
    - 甲、將修正節點改為指向其 parent
    - 乙、對改過之後的修正節點做左旋
    - 丙、將 Parent(即原修正節點的 Grandparent)塗成黑色
    - 丁、將 Grandparent 塗成紅色
    - 戊、對 Grandparent 做一次右旋
- iv. 修正節點的 parent 是右子：處理方式與上述對稱
  - 1. Uncle 是紅色
    - 甲、將 Uncle 及 Parent 塗成黑色
    - 乙、Grandparent 塗成紅色
    - 丙、將修正節點改為 Granparent
  - 2. Uncle 是黑色且修正節點為其 parent 的左子
    - 甲、將 Parent 塗成黑色
    - 乙、Grandparent 塗成紅色
    - 丙、對 Grandparent 做一次左旋
  - 3. Uncle 是黑色且修正節點為其 parent 的右子
    - 甲、將修正節點改為指向其 parent
    - 乙、對改過之後的修正節點做右旋
    - 丙、將 Parent(即原修正節點的 Grandparent)塗成黑色
    - 丁、將 Grandparent 塗成紅色
    - 戊、對 Grandparent 做一次左旋
- v. 重複判斷上述 6 個 case 並做出相應的修正，直到修正節點是根節點或修正節點的 parent 是黑色

#### D. Delete(函式命名為 delet，因 delete 為保留字)

- i. 需傳入要刪除的節點的值(NodeValue)
- ii. 用函式 find 尋找該節點位置，判斷該節點的左右子情形

1. 若左子或右子其中一個為 NIL 或兩個都為 NIL，將指標 y 指向該節點
2. 若左子及右子都不為空
  - 甲、尋找大於或等於該節點(該節點的右樹)中最小者作為替身，且替身的左子或右子至少有一個為 NIL
  - 乙、將 y 指向替身，將替身的值存入原刪除節點
- iii. 指標 x 指向 y 的孩子：經過上述 y 至多只有一個孩子
  1. 指標 x 的 parent 指向 y 的 parent
  2. 若 y 原為右子，將 y 的 parent 的右子指向 x
  3. 若 y 原為左子，將 y 的 parent 的左子指向 x
- iv. 若刪除的節點為黑色，則進入對 x 進行修正：因違反紅黑數規則，任一節點到後代的葉節點，需含同樣數量的黑色節點

#### E. Delete\_fixup(函式命名為 delet\_fixup)

- i. 修正節點為左子
  1. 若 sibling 是紅色
    - 甲、把 sibling 塗黑，parent 塗紅
    - 乙、對 parent 做一次左旋
    - 丙、進入後三個 case 其中之一：因仍為滿足任一節點到後代的葉節點，需含同樣數量的黑色節點
  2. 若 sibling 的左右子皆為黑色
    - 甲、把 sibling 塗紅
    - 乙、修正節點改為 parent
  3. 若 sibling 的右子為黑色，左子為紅色
    - 甲、把 sibling 左子塗黑，sibling 塗紅
    - 乙、對 sibling 做一次右旋
    - 丙、進入 case 4
  4. 若 sibling 的右子為紅色，左子為黑色
    - 甲、把 sibling 塗成跟 parent 一樣的顏色
    - 乙、Parent 塗成黑色
    - 丙、Sibling 的右子塗成黑色
    - 丁、對 parent 做一次左旋
    - 戊、將修正節點設為根節點：因 case 4 必會完成修正，故設為根節點跳出迴圈
- ii. 修正節點為右子
  1. 若 sibling 是紅色
    - 甲、把 sibling 塗黑，parent 塗紅
    - 乙、對 parent 做一次右旋

- 丙、進入後三個 case 其中之一：因仍為滿足任一節點到後代的葉節點，需含同樣數量的黑色節點
  - 2. 若 sibling 的左右子皆為黑色
    - 甲、把 sibling 塗紅
    - 乙、修正節點改為 parent
  - 3. 若 sibling 的左子為黑色，右子為紅色
    - 甲、把 sibling 右子塗黑，sibling 塗紅
    - 乙、對 sibling 做一次左旋
    - 丙、進入 case 4
  - 4. 若 sibling 的左子為紅色，右子為黑色
    - 甲、把 sibling 塗成跟 parent 一樣的顏色
    - 乙、Parent 塗成黑色
    - 丙、Sibling 的左子塗成黑色
    - 丁、對 parent 做一次右旋
    - 戊、將修正節點設為根節點：因 case 4 必會完成修正，故設為根節點跳出迴圈
- iii. 重複判斷上述 8 個 case 並做出相應的修正，直到修正節點是根節點或修正節點是紅色

#### F. LeftRotation

- i. 需傳入要左旋的節點
- ii. 將該節點的右子改為原右子的左子，且將新右子的 parent 指向自己
- iii. 原右子的 parent 改為指向自己的 parent
- iv. 若自己不是根節點
  - 1. 若自己原為右子，將自己的 parent 的右子指向自己的原右子
  - 2. 若自己原為左子，將自己的 parent 的左子指向自己的原右子
- v. 若自己是根節點，將 root 改為指向原右子
- vi. 將原右子的左子指向自己，並將自己的 parent 指向原右子

#### G. RightRotation

- i. 需傳入要右旋的節點
- ii. 將該節點的左子改為原左子的右子，且將新左子的 parent 指向自己
- iii. 原左子的 parent 改為指向自己的 parent
- iv. 若自己不是根節點
  - 1. 若自己原為右子，將自己的 parent 的右子指向自己的原左子
  - 2. 若自己原為左子，將自己的 parent 的左子指向自己的原左子

- v. 若自己是根節點，將 root 改為指向原左子
- vi. 將原左子的右子指向自己，並將自己的 parent 指向原左子

#### H. Find

- i. 需傳入要尋找的節點的值(NodeValue)
- ii. 自 root 開始比較
- iii. 若要尋找的節點的值小於該節點，即繼續與該節點的左子比較
- iv. 若要尋找的節點的值大於或等於該點，即繼續與該節點的右子比較
- v. 不斷重複上述動作，直至要尋找的節點的值與該節點相同

#### I. Inorder

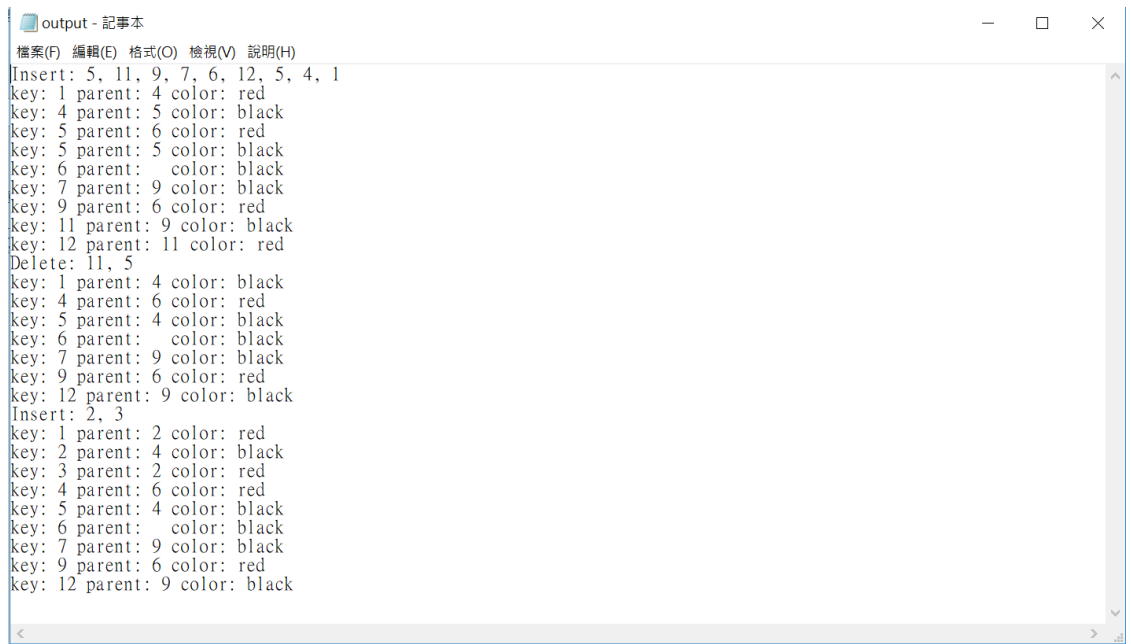
- i. 採用遞迴方式
- ii. 呼叫自己走訪左子
- iii. 印出走訪的點的值、parent 及顏色
- iv. 呼叫自己走訪右子
- v. 根據上述步驟，會先拜訪最左邊的節點，而後是該點的 parent，最後是自己的 sibling(即 parent 的右子)，重複上述步驟直到走訪全樹，符合 inorder(leftchild→parent→rightchild)

### 3. Result

#### A. Dev C++ 編譯並執行後的結果

```
C:\Users\user\Desktop\演算法\作業\RBtree\RBTree.exe
key: 4 parent: 5 color: black
key: 5 parent: 6 color: red
key: 5 parent: 5 color: black
key: 6 parent:  color: black
key: 7 parent: 9 color: black
key: 9 parent: 6 color: red
key: 11 parent: 9 color: black
key: 12 parent: 11 color: red
Delete: 11, 5
key: 1 parent: 4 color: black
key: 4 parent: 6 color: red
key: 5 parent: 4 color: black
key: 6 parent:  color: black
key: 7 parent: 9 color: black
key: 9 parent: 6 color: red
key: 12 parent: 9 color: black
Insert: 2, 3
key: 1 parent: 2 color: red
key: 2 parent: 4 color: black
key: 3 parent: 2 color: red
key: 4 parent: 6 color: red
key: 5 parent: 4 color: black
key: 6 parent:  color: black
key: 7 parent: 9 color: black
key: 9 parent: 6 color: red
key: 12 parent: 9 color: black
-----
Process exited after 0.08909 seconds with return value 0
請按任意鍵繼續 . . .
```

## B. Output.txt



```
output - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)
Insert: 5, 11, 9, 7, 6, 12, 5, 4, 1
key: 1 parent: 4 color: red
key: 4 parent: 5 color: black
key: 5 parent: 6 color: red
key: 5 parent: 5 color: black
key: 6 parent:  color: black
key: 7 parent: 9 color: black
key: 9 parent: 6 color: red
key: 11 parent: 9 color: black
key: 12 parent: 11 color: red
Delete: 11, 5
key: 1 parent: 4 color: black
key: 4 parent: 6 color: red
key: 5 parent: 4 color: black
key: 6 parent:  color: black
key: 7 parent: 9 color: black
key: 9 parent: 6 color: red
key: 12 parent: 9 color: black
Insert: 2, 3
key: 1 parent: 2 color: red
key: 2 parent: 4 color: black
key: 3 parent: 2 color: red
key: 4 parent: 6 color: red
key: 5 parent: 4 color: black
key: 6 parent:  color: black
key: 7 parent: 9 color: black
key: 9 parent: 6 color: red
key: 12 parent: 9 color: black
```